UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño
y programación de Videojuegos

UNLVIRTUAL

Introducción a la programación

Unidad Temática 1
Preparando las herramientas

- **Objetivo:** interiorizarse en los conceptos básicos relativos a la creación de programas y comprender un típico entorno de desarrollo en C++.

- **Temas:** Introducción. Conceptos básicos: Algoritmo, Programa, Programación. Resolución de Problemas.
Lenguaje de programación. Compilador. Depuración.
Instalación y uso del compilador (IDE DevC++).
Creando un espacio de trabajo.
Estableciendo un estándar en el desarrollo.

INTRODUCCIÓN

Como habrán leído en la presentación a la materia, vamos a introducirnos en la programación a través de un lenguaje llamado C. Programar es decirle a la computadora qué es lo que queremos que haga, lo cual no es tan simple, porque además hay que decirle cómo tiene que hacerlo y, para agregar dificultad, se lo tenemos que decir en los términos que ella entienda.

Supongamos que tiene que indicarle a una persona cómo ir a la terminal de colectivos. De repente nos damos cuenta de que no habla castellano, sólo entiende chino mandarín, por lo que debemos encontrar un lenguaje común a ambos. Como nuestro interlocutor sólo entiende chino mandarín, ningún otro idioma (como el inglés por ejemplo) sirve. Finalmente podríamos usar un sistema básico de señas, o bien dibujar.

El C es un lenguaje intermedio entre el humano y el de la máquina. Sin embargo hay un asunto que en nuestro ejemplo pasamos muy fácilmente por alto, porque generalmente lo consideramos muy obvio. Para poder indicar a otra persona cómo llegar a la terminal de colectivos, nosotros debemos saber primero cómo llegar a esa terminal de colectivos. Luego debemos transmitírselo a la otra persona, y esta transmisión lleva sus dificultades. Esto será abordado en las unidades subsiguientes, en esta unidad trataremos el primer problema: saber primero cómo hacer algo. Antes de decirle a la computadora que hacer y cómo, somos nosotros quienes tenemos que saber qué se debe hacer y cómo.

Supongamos que nuestro amigo perdido sí habla nuestro idioma. Como recorremos el mismo camino todos los días le puedo decir:

seguí derecho 2 cuadras,
doblá a la izquierda
y seguí adelante otras 4 cuadras,
ahí buscá una pequeña cortada que te deja en la puerta de la terminal

Esta serie de instrucciones que acabamos de dar entra en lo que se daría a llamar un *algoritmo.*

> Un algoritmo es un conjunto finito de operaciones (instrucciones, pasos) que seguidos en un determinado orden permiten resolver un tipo de problema.

Sin embargo estamos pasando por alto otro detalle dado por obvio: ¿qué pasaría si nosotros nunca caminamos por ese lado de la ciudad? En ese caso podemos ir a un kiosco cercano, comprar un mapa y, mirando el mapa, plantear las posibles rutas para llegar a la terminal. Entonces, antes de crear nuestro algoritmo para dirigir a nuestro amigo hacia la terminal, estamos planteando una estrategia para poder crear nuestro algoritmo.

Nuestra estrategia es: ir a un kiosco comprar un mapa, anticipar el recorrido de las posibles rutas para llegar a destino, sopesar los pros y las contras de cada una y decidirnos por una solución que a nuestro entender es la mejor y con ella crear nuestro algoritmo.

En palabras más sencillas, decidir si primero ir derecho y después doblar, o primero doblar y después ir derecho…, uno puede valorar la mejor ruta porque es de noche y está mejor iluminada, o porque una es más tranquila y

en la otra hay mucho tránsito, o por el motivo que fuera, dependiendo de la situación una puede ser mejor que la otra.

La palabra clave de todo este proceso es: **ESTRATEGIA**

Estrategia: no importa qué tan obvio sea, nunca se crea un algoritmo sin una estrategia previa porque:

- No existe un único algoritmo para resolver un único problema, la conveniencia entre uno y otro depende de factores que varían de una situación a otra.

- Fundamentalmente, antes de darle indicaciones a alguien tenemos, forzosamente, que saber cuál es el resultado final y cómo llegar a él. En nuestro ejemplo, si no recorrimos las rutas efectivamente, al menos debimos hacerlo virtualmente a través del mapa, para asegurarnos que nuestro algoritmo efectivamente lleve a la solución.

Así que la clave para resolver un problema mediante un algoritmo es:

- **Pensar** primero una estrategia
- codificar el algoritmo sobre la base de la estrategia
- y si no sabemos cómo, conseguir la información necesaria (en nuestro ejemplo, nos apoyamos en un mapa de la ciudad).

Si el ejemplo hubiese sido salir de un laberinto, podría poner el mismo algoritmo, si la salida está en la misma posición que la terminal de colectivos:

seguí derecho 2 bloques,
doblá a la izquierda
y seguí adelante otros 4 bloques,
ahí buscá una pequeña cortada que te deja en la puerta del laberinto.

Los juegos de laberinto son muy comunes. Basta mirar una revista y en la parte de entretenimiento suele haber laberintos para resolver. De hecho en computación, resolver problemas de laberinto es algo muy desarrollado, ya que es análogo a ir de un punto a otro esquivando los obstáculos. En el caso de los laberintos, las paredes son los obstáculos.

Nuestro algoritmo anterior funciona bien si conocemos el mapa del laberinto. Pero si nos vendan los ojos y nos dejan en medio de uno: ¿existirá un algoritmo genérico que resuelva nuestro problema de salir?

Algoritmo de Tremaux
1. No siga el mismo camino dos veces.
2. Si llega a un cruce nuevo, no importa qué camino siga.
3. Si un camino nuevo lo lleva a un cruce viejo, o a un callejón sin salida, retroceda hasta la entrada del camino.
4. Si un camino viejo lo lleva a un cruce viejo, tome un camino nuevo, y si no lo hay, tome cualquiera.

¿Este será el único algoritmo que existe? No, está por ejemplo el de la mano derecha o izquierda, según el gusto del usuario y seguramente debe haber más.

Nos detenemos aquí para resolver una práctica. Ejercitemos el pensamiento (práctica). El ejercicio consiste en discutir las estrategias para fomentar el pensamiento.

Adivina un número

Dos personas: la primera piensa un número y la segunda intenta adivinarlo. Cada vez que la segunda dice un número la primera le da una pista si el número que dijo es más alto o más bajo que el número que pensó. Plantear un algoritmo para resolverlo. (Variación: frío/caliente, si el número se encuentra más lejos o más cerca que el número dicho anteriormente; templado si está a la misma distancia)

Acertar el blanco

Un cañón dispara a un objetivo. El cañón puede graduar el ángulo de disparo, quien dispara posee unos binoculares que le permiten ver dónde cae la bala. Plantear un algoritmo para dar en el blanco.

Tres en línea (en inglés "tic tac toe")

Un algoritmo para jugar y, si es posible, ganar siempre. Este juego tiene una particularidad: con una buena estrategia no se puede perder. Si no conocen como ahí esta Google.

La batalla naval

El juego de la batalla naval, ¿se puede aplicar un algoritmo para jugarlo? Antes de escribir un algoritmo, plantear una estrategia.

Algoritmo para resolver sopa de letras. Antes de resolver el algoritmo plantear una estrategia.

Diseño Descendente o Top Down

A dos investigadores de IBM se le ocurrió que se podía tener una estrategia general para enfrentar todos los problemas, y diseño lo que se conoce como estrategia top down. [Consultar: http://es.wikipedia.org/wiki/Top-down_y_Bottom-up) En simples palabras, consiste en ir dividiendo el problema en subproblemas más sencillos hasta un nivel en que el algoritmo que se resuelve el pequeño problema sea simple. Entonces la solución total es la suma de todas las soluciones simples.

Veamos un problema simple (pero un poquito más complejo):
seguir derecho 2 cuadras,
doblar a la izquierda
y seguir adelante otras 4 cuadras,
ahí buscar una pequeña cortada que nos deja en la puerta de la terminal

Como el problema es muy simple lo podemos dividir en sólo dos subproblemas:
llegar hasta la zona de la cortada
buscar la cortada

llegar hasta la zona de la cortada:
seguir derecho 2 cuadras,
doblar a la izquierda
y seguir adelante otras 4 cuadras

buscar la cortada:
comenzar en una esquina de la cuadra
caminar hasta que una de las veredas se interrumpa por una bocacalle
adentrarse en la bocacalle hasta la terminal

En la práctica se plantea en forma general una solución mediante acciones que todavía no sabemos cómo se van a resolver, pero que suponemos que resuelven el problema. Por ejemplo **buscar la cortada:** cuando lo planteamos suponemos que existe una serie de acciones que resolverán ese problema, y más adelante nos enfocamos sólo en resolver buscar la cortada.
Ahora veamos un problema lo suficientemente complejo como para que no quede más alternativa que dividirlo en partes más simples.

El general
Estás a cargo de 2 tropas de infantería, 1 unidad de transporte mecanizada que puede transportar 1 unidad de tropas de infantería o abastecimiento, 1 unidad de tanques, 1 unidad de artillería, las tropas de infantería necesitan

descanso y comida, las unidades mecanizadas necesitan combustible, y todos necesitan municiones. La salud de cada unidad depende del impacto del enemigo sobre ella, y la efectividad de las unidades depende de si tienen suficiente munición, y de la moral que a su vez depende de las posibilidades de ganar la batalla que ven las tropas, si son muy dañadas la moral cae, si la infantería no recibe descanso y comida también cae, si una unidad es destruida la moral de las otras también cae. El enemigo tiene la misma cantidad y tipo de tropas. Realizar un algoritmo para ganar la batalla.

En este ejercicio lo importante es pensar una estrategia general, para luego pensar las alternativas, es lo suficientemente amplio que no tiene una solución única.

Estrategia principal
El abastecimiento es la clave. Si las unidades se quedan sin combustible no tienen movilidad, si se quedan sin municiones pierden su efectividad de ataque, si el alimento no llega las tropas de infantería también pierden su efectividad.
Para impedir el abastecimiento, el objetivo principal debe ser la unidad de transporte. Si la unidad de transporte es destruida, todas las demás unidades se resentirán.
Esta estrategia es tan válida como cualquiera, otra estrategia puede basarse en el hostigamiento.

Tácticas
siempre atacar en superioridad numérica
siempre proteger nuestra unidad de transporte
ante inferioridad numérica retroceder a posición segura
buscar la posibilidad de flanquear

Algoritmo principal

si hay **posibilidad de atacar** unidad de transporte enemiga
    si hay posibilidad de victoria

        atacar
    sino
        si hay **posibilidad de atacar** cualquier unidad enemiga
            si hay posibilidad de victoria
                atacar
moverse para **mejorar posibilidad de victoria**

Como podemos ver, en el algoritmo principal hemos planteado las acciones en forma general, suponiendo que existen algoritmos que las resuelven. Ahora atacaremos cada problema por separado:

Posibilidad de atacar

Si la unidad enemiga se encuentra al alcance de una unidad nuestra
o unidad enemiga se encuentra aislada en inferioridad numérica con respecto a un grupo de nuestras tropas de tal manera que lleguen antes a su posición que las demás unidades enemigas
    entonces Posibilidad de atacar es verdadera
sino
    Posibilidad de atacar es falsa

Atacar

Si se está a distancia entonces disparar
Sino
    moverse para llegar a alcance cuidando de no quedar aislado y atacar

Mejorar posibilidad de victoria

Implementar tácticas para aislar tropas y mantener líneas de abastecimiento (es un buen top down para desarrollar). Acá sólo se desarrollan algunos problemas para ilustrar que se puede seguir subdividiendo en problemas menores.

Si es posible flanquear al enemigo y no compromete nuestro abastecimiento
Flanquear
Sino
Si es posible hostigar al enemigo y no compromete nuestro abastecimiento
Hostigar
si el enemigo tiene posibilidades de flanquearnos
retroceder unidades flanqueables a posiciones cerradas
sino
si unidades enemigas hostigan a una unidad nuestra
retroceder unidad hostigada o avanzar unidades para fortalecer la línea tratando de mantener línea de abastecimiento

Flanquear (acción de rodear a una unidad enemiga de tal manera de atacar por su punto débil, normalmente un rente de combate se caracteriza por formar una línea imaginaria entre todas las unidades, esa línea contiene una unidad al comienzo y otra al final, flanquear la línea es rodear de tal manera de atacar a una de estas dos unidades por el lado, de tal manera que las otras no puedan protegerla)

mover dos unidades o más hacia una de las puntas de la línea enemiga
rodear la punta y atacar a la unidad más expuesta
si las tropas enemigas no cierran una nueva línea atacar la siguiente.

Hostigar (acción de atacar a una tropa más poderosa y lenta de reacción con una más débil y rápida, de tal manera de suspender el ataque y huir al momento que la tropa atacada reaccione para el contraataque, y siendo más rápida lograrse poner a salvo fuera del alcance de la lenta)

aproximarse a distancia de tiro
atacar por breve período de tiempo
retroceder hasta posición segura
repetir hasta que el hostigado sea destruido o se coloque en posición que ya no pueda ser hostigado

En nuestro ejemplo no están desarrolladas todas las posibilidades, pero con esto creemos que queda una idea del diseño top down en el sentido de ir descomponiendo en problemas más simples.

Ahora definamos formalmente y profundicemos el concepto de estrategia y de algoritmo.

> ESTRATEGIA
> El diseño de la estrategia consiste en encontrar un método que nos permita llegar a resolver el problema planteado. Como primer paso de esta etapa, debemos preparar un plan o esquema general de las tareas que deben realizarse para llegar a la solución.
> Este esquema se denomina estrategia y debe ser una lista de qué hacer.

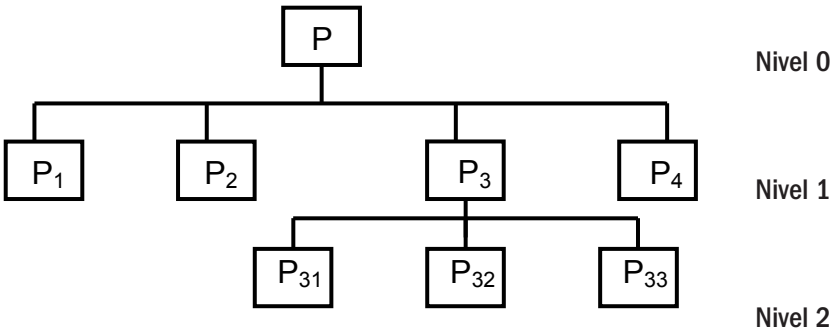
¿Cómo se diseña una estrategia?

Por lo dicho, diseñar una estrategia consiste en dividir o descomponer el problema original en una sucesión de problemas más simples, de tamaño suficientemente pequeño como para que cada uno de ellos pueda ser comprendido en su totalidad. Esto permitirá atacar la solución de cada problema

simple por separado e independientemente de los demás, volviendo a aplicar este enfoque a cada uno de los subproblemas hasta llegar a subproblemas de solución simple. Una vez que todos ellos han sido resueltos, se puede decir que el problema original ha sido resuelto.

Este proceso de descomposición de un problema partiendo de la formulación completa del problema hasta llegar a problemas elementales de simple solución, se llama diseño descendente, también conocido como top-down, método de refinamiento sucesivo o diseño compuesto.
Gráficamente, dado el problema P lo dividiremos en subproblemas Pi. Cada subdivisión implica un descenso de nivel.

Siguiendo nuestro ejemplo de la batalla



P = Algoritmo Principal

P_1 = Posibilidad de Atacar
P_2 = Atacar
P_3 = Mejorar posibilidad de Victoria
P_4 = Posibilidad de Victoria

P^{31} = Flanquear
P^{32} = Hostigar
P_{33} = Retroceder

Cada P_i representa un enunciado o subproblema. Para cada uno existen 2 posibilidades:

- que P_i sea un subproblema o una tarea simple, dando por finalizada la descomposición
- que P_i sea un subproblema o una tarea compuesta y por lo tanto sea posible su descomposición en una nueva secuencia de subproblemas

Las características generales de este tipo de diseño se basan en:
- ir de lo general a lo particular
- no existe una única descomposición de subproblemas
- en cada nivel puede verificarse que el esquema sea el correcto

Finalmente se realiza un trabajo de recomposición del esquema completo, resolviendo cada subproblema hasta lograr la solución del problema. El diseño de una estrategia y su posterior refinamiento, constituyen las etapas más creativas y quizás más dificultosas de todo el proceso de resolución de un problema.

Planteada una estrategia indicando qué tareas hacer, debemos especificar una lista detallada de *cómo* hacerlas, llegando así a definir una solución paso a paso del problema llamada algoritmo. La descripción de la solución detallada por medio de un algoritmo constituye el segundo paso en la etapa de elección del método.
La palabra algoritmo se utiliza, en general, como sinónimo de procedimiento, método o técnica. Pero en el área de computación tiene un significado más específico.

ALGORITMO
Un algoritmo es un conjunto finito de operaciones (instrucciones, pasos) que seguidos en un determinado orden permiten resolver un tipo de problema.

Las características principales de un algoritmo son:
- Finito: permite arribar a la solución de un problema después de la ejecución de un número finito de pasos.
- Definido: cada paso debe ser enunciado en forma clara y precisa, y no debe dar lugar a ambigüedades. Para los mismos datos el algoritmo debe dar siempre los mismos resultados
- General: la solución debe ser aplicable a un tipo de problemas y no a un problema particular.

Teniendo en cuenta las características mencionadas previamente podemos agregar que:

Un algoritmo es una secuencia ordenada y finita de pasos que constituyen un método general para resolver un tipo de problemas.

Es de notar que esta definición, se refiere a "resolver un tipo de problemas" y no hace hincapié en el uso del computador como herramienta para su resolución. Esto se debe a que el concepto de algoritmo se aplica a problemas computacionales que van a ser resueltos por medio de un computador y a problemas no computacionales, en cuya resolución no interviene esta herramienta. En ambos casos el lenguaje usado en la descripción del algoritmo debe ser comprensible para el destinatario o para quien lo va a ejecutar. Por lo visto, para cualquier problema para el que pueda especificarse un método finito de solución puede definirse un algoritmo.

Ejemplos que se pueden presentar en la vida diaria:
- una receta de cocina
- las instrucciones para utilizar un aparato electrónico
- el camino para llegar a un lugar determinado desde un punto de partida

Ejemplos de algoritmos computacionales:
- Calcular los sueldos de los empleados de una empresa
- Actualizar el stock de un comercio
- Calcular las raíces de una ecuación

Desarrollemos el siguiente ejemplo de la vida diaria

Problema: Preparar un taza de café instantáneo
El grado de detalle que deberemos usar en la definición del método, dependerá de la persona que sea la ejecutante de la solución. Si el ejecutante es un ama de casa, probablemente con el enunciado sea suficiente, pero si se trata de alguien que nunca preparó un café podríamos detallar los siguientes pasos:

PROCESO Cafe1
- Calentar una taza de agua sin llegar al punto del hervor;
- Poner en un taza tres cucharaditas de azúcar, dos de café instantáneo y media cucharadita de soda;
- Batir hasta que la mezcla se torne marrón claro;
- Llenar con el agua caliente la taza;
- Revolver para disolver la mezcla en el agua

FINPROCESO

Obsérvese que para indicar el inicio y el fin del algoritmo se han utilizado las palabras **PROCESO** y **FINPROCESO** respectivamente, y que los pasos han sido lo suficientemente simples para un principiante en el arte de preparar café.
Otro aspecto que es importante considerar es que contamos con una serie de elementos para poder preparar el café: recipiente para calentar el agua,

azúcar, café, cucharita, taza, soda.
Supongamos que no se tiene la certeza de que en el momento de hacer el café se tenga soda, por ende, este elemento se podrá reemplazar con agua, con lo cual el algoritmo será:

PROCESO Cafe2
- Calentar una taza de agua sin llegar al punto del hervor;
- Poner en un taza tres cucharaditas de azúcar, dos de café instantáneo
- **SI** se tiene soda
  ENTONCES agregar en la taza media cucharadita de soda
  SINO agregar en la taza media cucharadita de agua fría
- **FINSI**
- Batir hasta que la mezcla se torne marrón claro;
- Llenar con el agua caliente la taza;
- Revolver para disolver la mezcla en el agua

FINPROCESO

Las dos primeras instrucciones se ejecutan una a continuación de otra. Luego se presentan dos alternativas: o se agrega media cucharadita de soda o se agrega media cucharadita de agua fría. Para describirlas se ha usado las palabras **SI**, **ENTONCES**, **SINO**, **FINSI**, que se analizarán en detalle más adelante.

En este caso el algoritmo cubre ya mayor cantidad de posibilidades, no previstas en la versión anterior. A partir de esto se pueden realizar las siguientes observaciones:

- El algoritmo debe estar compuesto por acciones tales que el ejecutante sea capaz de realizar.
- El algoritmo debe ser enunciado en un lenguaje comprensible para el ejecutante, hombre o computador. En este último caso, estará restringido a un juego de instrucciones perfectamente determinado.
- El algoritmo deberá representar todo el conjunto de posibles resultados del problema, inclusive el caso de que no tenga solución.
- Para un mismo problema se puede describir más de un algoritmo, y con cualquiera de ellos se deberá llegar a la/s misma/s solución/es; un algoritmo será más eficaz que otro. La eficacia del algoritmo depende de los recursos con que se cuente y los factores que se consideren: costos, tiempo, etc.

Ejemplo de resolución de problemas de la vida cotidiana

Ejemplo: Preparar un licuado de frutas
Recursos: Licuadora, fruta con cáscara, taza con leche, taza con azúcar, cuchillo, plato. Todos los elementos están sobre la mesada. Se cuenta con las medidas necesarias de todos los ingredientes.
Algoritmo:

PROCESO Licuado
- Tomar el vaso de la licuadora
- Colocar el vaso en la base
- **SI** la licuadora no está enchufada
ENTONCES enchufarla
- **FINSI**
- Tomar el cuchillo
- **REPETIR**
tomar la fruta
pelarla
cortarla sobre el plato
colocar la fruta cortada en el vaso
- **HASTAQUE** no haya más frutas
- Dejar el cuchillo
- Tomar la taza con la leche
- Echar la leche en el vaso
- Dejar la taza de la leche sobre la mesada
- Tomar la taza con el azúcar
- Colocar el azúcar en el vaso
- Dejar la taza del azúcar sobre la mesada
- Tapar el vaso
- Mover la perilla de encendido hacia la derecha
- **REPETIR**
esperar
- **HASTAQUE** la mezcla esté licuada
- Mover la perilla de encendido hacia la izquierda
FINPROCESO

En este caso ciertas acciones como la de Tomar la fruta, pelarla, cortarla, colocar… se repiten mientras se tiene fruta para hacerlo. Aparecen aquí las palabras **REPETIR y HASTAQUE**, que veremos en capítulos posteriores.

RESOLUCIÓN DE PROBLEMAS COMPUTACIONALES

En la vida diaria nos enfrentamos continuamente a problemas que debemos resolver en lo posible felizmente. Así como cada individuo tiene formas de encarar un problema y su propia manera de solucionarlo, computacionalmente hablando podemos hacer un paralelo. Ante la presentación de un problema, encarar la mejor forma de resolverlo para arribar al resultado esperado y correcto es un desafío. Para ello debemos comprender exactamente qué se pide, qué resultados se pretenden y qué restricciones y/o condiciones existen. Para realizar lo antes dicho dividiremos la resolución de un problema en etapas, las cuales enunciamos y definimos a continuación.

Etapas para la resolución de problemas

a. Definición del problema
Está dada por la formulación del problema en forma correcta y completa. Esta enunciación de lo que se desea es primordial para el éxito de la resolución.

b. Análisis del problema
A partir del estudio del problema se deberán identificar y conocer las partes principales del mismo y, de esa manera, determinar los siguientes conjuntos:
- de datos: es la información con que contamos para resolver el problema;
- de resultados: es lo que se desea obtener;
- de condiciones: una o más relaciones que vinculan los dos conjuntos anteriores y que permitirán plantear la solución del problema.

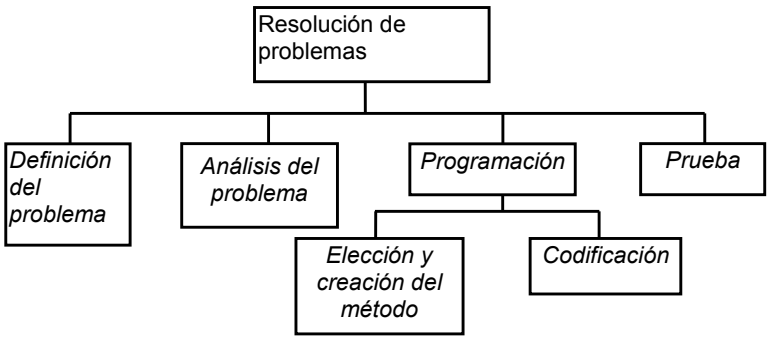
c. Programación
Esta etapa consiste en obtener la solución del problema dado. Se divide en dos subetapas:

c.1. Elección y creación del método
Se trata de buscar un procedimiento o método general que permita resolver el problema planteado utilizando una computadora. Es muy factible que se encuentren varios métodos para hacerlo, lo importante es determinar la "mejor alternativa", de acuerdo a distintos parámetros que se establezcan para esta selección. Esta puede ser la que produzca los resultados esperados en el menor tiempo y al menor costo, o sólo en el menor tiempo u otras.

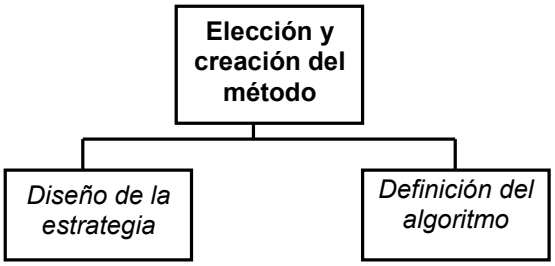
c.2. Codificación
Consiste en expresar el método elegido en un lenguaje, llamado Lenguaje de programación, que pueda ser interpretado por la computadora. Esta subetapa será objeto de estudio en años superiores de la carrera.

d. Prueba
Esta etapa consiste en la ejecución del código del método elegido, es decir, suministrar los datos al computador, y obtener los resultados. Luego se analizarán los mismos determinando si son realmente los esperados. Caso contrario, deberán analizarse las etapas previas, comenzando por la última hacia atrás, y realizar las modificaciones necesarias, repitiendo esta proceso hasta obtener los resultados esperados.

Observemos gráficamente las etapas descriptas.



La etapa de elección y creación del método se puede dividir a su vez en el *Diseño de la estrategia y la Definición del algoritmo,* y puede graficarse de la siguiente manera:



FORMALIZACIÓN

Formalizaremos algunos conceptos vistos anteriormente.

Hemos mencionado que la forma de enunciar la solución a un problema planteado depende del *ejecutante* o también llamado *procesador.* Por lo tanto llamaremos así a toda entidad capaz de entender un enunciado y ejecutar los pasos descriptos en un algoritmo. Si bien en los ejemplos vistos el ejecutante se trataba de una persona en la resolución de problemas computacionales debemos pensar que el procesador será la computadora.

También hemos notado que para poder realizar su tarea, el ejecutante debe contar con los recursos adecuados. El conjunto de estos recursos existentes en el momento de la ejecución de un trabajo constituye el *ambiente* del problema.

El método que se elija para proponer la solución de un tipo de problema depende del ejecutante y de los recursos o elementos con que se cuenta (ambiente). Cuando definimos algoritmo hemos hablado de un conjunto de pasos o acciones.

Una acción es un evento que modifica el ambiente y puede ser:
- Primitiva
- No-primitiva

Una acción es **primitiva** cuando para un ejecutante dado su enunciado es suficiente para que pueda ser ejecutada sin información adicional.
Una acción **no-primitiva** es aquella que puede ser descompuesta en acciones primitivas para un ejecutante dado.

También hemos visto que en los ejemplos se nos presentan situaciones que indican alternativas: “SI se tiene soda...”. Esta no es una acción porque no modifica el ambiente, pero son elementos que el ejecutante debe saber interpretar. A estos enunciados se los denomina condición. Por lo tanto:

Una **condición** es una afirmación lógica sobre el estado de algún recurso del ambiente, que puede tomar valor verdadero o falso en el momento de la observación.

El ejecutante determina las acciones a seguir en el momento de la ejecución del algoritmo, dependiendo de que la condición sea satisfecha o no.

PROGRAMACIÓN MODULAR

Es un método de diseño y tiende a dividir el problema en partes perfectamente diferenciadas que puedan ser analizadas, resueltas y puestas a punto por separado.

Para atacar el análisis de un problema, y siguiendo el diseño *top-down*, se pueden utilizar criterios de programación modular para dividirlos en partes independientes, probando cada uno por separado y realizando su recomposición ascendente.

Cada una de las partes independientes se llama Módulo y para su determinación se deben tener en cuenta los siguientes criterios:

- un módulo debe corresponder a una función lógica perfectamente bien definida;
- los módulos deben ser pequeños para que sean claros y de poca complejidad;
- un módulo debe tener una estructura de caja negra, es decir la salida debe ser exclusivamente función de la entrada;
- cada módulo deber tener una única entrada y una única salida.

Objetivos de la programación modular

La programación modular tiende a:
- *disminuir complejidad:* disminuye la complejidad del problema original, dividiendo un problema en partes más simples.
- *aumentar la claridad:* el problema original es planteado ahora como una sucesión de módulos que resulta más fácil de comprender inclusive para terceras personas.
- *aumentar la fiabilidad:* como consecuencia de los dos puntos anteriores, aumenta la confiabilidad en todo proceso de resolución.
- *facilitar modificaciones y conexiones:* cada módulo puede realizarse y probarse por separado, minimizándose los problemas de puesta a punto al final.